An Evaluation Function for Othello Based on Statistics

Michael Buro
NEC Research Institute
4 Independence Way
Princeton, NJ 08540 USA

e-mail: mic@research.nj.nec.com

Abstract

This paper describes the evaluation function of one of today's strongest Othello programs — LOGISTELLO. The function is based on statistical analysis of a large set of example game positions and profits from ideas regarding pattern value estimation, determination of feature weights, and the fast approximation of time consuming evaluation features.

Introduction

Most game–playing programs use evaluation functions to estimate the players' winning chances at the leaves of game–trees. Normally, these functions combine features that measure specific properties of the position which are correlated with winning. In this paper such an evaluation function for the game of Othello[1] is described, which is almost entirely based on statistical analysis. It is the heart of LOGISTELLO, considered to be one of the strongest Othello programs ever.

Central to the following discussions is the use of a large set of loss–draw–win classified example positions. This serves several purposes:

- First, it is used to compute the *discordance* of features which is a global measure for the feature's ability to separate positions from different classes.[2] In comparison with the feature's misclassification rate, the discordance is more meaningful in the context of game–tree search since normally the goal is to find the best move rather than to decide whether a position is won or lost.
- Secondly, the large position set is used to evaluate instances of board patterns by estimating the winning chances conditioned upon their occurrence.
- Finally, feature weights are determined by fitting a statistical model.

Over a period of two years, about 60,000 Othello games[3] were played by early versions of Igor Đurđanović's program KITTY and LOGISTELLO. During the last two years, LOGISTELLO generated about 20,000 additional games while extending its opening book. Positions from these games are also used to increase the number of examples especially for pattern value estimation. All in all, a total of approximately three million game positions were classified by negamaxing the final game results in the tree built from all games.

The following sections describe LOGISTELLO's evaluation features, their combination, and implementation aspects.

Evaluation Features

LOGISTELLO's features fall into two classes, namely

- Mobility measures
- Patterns

which together approximate important concepts in Othello, like striving for stable discs, maximizing the number of moves, and parity.

Mobility

Besides the ownership of stable discs (that cannot be flipped by the opponent), mobility plays a central role in Othello. Many mobility measures have been discussed in the literature (Rosenbloom 1982; Mitchell 1984). The simplest approach is to count legal moves but — unfortunately — the determination of all legal moves is very time consuming. Early versions of LOGISTELLO used about 50% of their time for this task. Nowadays, LOGISTELLO uses a fast approximation of the simple mobility measure. Here the idea is to approximate the globally defined mobility by the sum of mobilities local to the lines of the board, i.e. the horizontals, verticals, and diagonals, in order to avoid *any* legal–move computation.[4] Diagonals of

[1] For those readers unfamiliar with this game, a brief description is given in the appendix.

[2] To be precise, the discordance is defined as the ratio of the number of incorrectly separated pairs of positions with different outcomes and the total number of all such pairs.

[3] The game file can be obtained via anonymous ftp: `ftp.nj.nec.com/pub/igord/othello/misc/database.zip`

[4] Lee & Mahajan (1990) used this technique to adjust the *exact* mobility score.

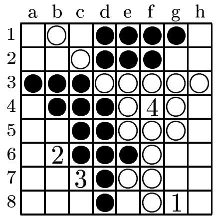


Figure 1: Moves that are counted k–times by the mobility approximation ($1 \le k \le 4$).

length three are not considered, since the corresponding moves flip discs on X–squares which is dangerous. Moreover, omitting these diagonals speeds up the incremental update of the pattern indices as discussed in the implementation section. In this way the mobility can be quickly approximated by simply adding 34 values that are stored in tables. However, it must be asked, whether the new mobility measure can compete with the original exact mobility, since there are situations in which it overestimates the number of moves (Figure 1). As Figure 4 shows, in the opening and middle–game the exact mobility has only a slightly better separation ability than its approximation, whereas in the endgame this advantage vanishes. Thus, the exact mobility can be safely replaced by its much faster approximation.

Potential Mobility

Since current mobility is an important feature in Othello, it is a good strategy to strive for positions in which future move possibilities are likely. For making a move, it is necessary that the move square is empty and there is an adjacent opponent's disc which can be bracketed. Thus, the relationship between empty squares around opponent's discs is important for future mobility. Rosenbloom (1982) proposed three measures for potential mobility. Since these are strongly correlated, only the feature with the lowest test–set discordance among the three was chosen and slightly adapted: LOGISTELLO defines the potential mobility for one player as the number of adjacent pairs of opponent's discs and empty squares. Only those empty squares are considered which might be used to flip an opponent's disc, but which — in order to decrease the correlation of potential and current mobility — are not currently legal moves. Furthermore, potential mobility on edges and length–3 diagonals are ignored, since placing a disc on these lines might generate legal moves in the future but is otherwise very dangerous because corners and X–squares are probably involved. Finally, the difference of the players' values can be approximated analogously to current mobility by a sum of line values in order to speed up the computation.

Patterns

There are many edge situations which lead to a forced win or loss of a corner. Therefore, it is important to be aware of them when evaluating a leaf position in the game tree. In early approaches values were manually assigned to edge instances and stored in a table. This allows a fast edge evaluation since only four values have to be added. Rosenbloom (1982) and Lee & Mahajan (1990) proposed semi–automatic (probabilistic) minimax procedures for edge–table generation which worked quite well. However, there is a problem with these approaches: for the recursive edge filling process, many constants had to be pre–defined intuitively. This raises the question whether the quality of the computed table could be increased by altering the constants.

A New Approach In order to circumvent this problem, and to make the pattern approach more flexible, the early concepts can be generalized as follows:

- Any set of squares is called a *pattern*.
- A pattern *instance* (or *configuration*) assigns an element from {empty, black, white} to each pattern square. Without loss of generality, it is assumed that Black is to move.
- The *value* of a pattern instance is defined as the expected value of a suitable random–variable — defined on the set of all positions — conditioned upon the occurrence of that instance.

In order to illustrate this definition, Figure 2 shows all patterns that are currently used by LOGISTELLO, and Figure 3 gives some edge configuration examples. Meaningful definitions for the random–variable include the game outcome (win= 0.5, draw= 0, loss= −0.5) or the final disc–difference (−64..64) from Black's point of view after optimal play.

For some patterns — especially for those including corners — the side to move makes a big difference and one might guess that it is very important to distinguish both cases. However, the right to move can only be realized in one region of the board, and optimizing this aspect everywhere can lead to an overestimation of

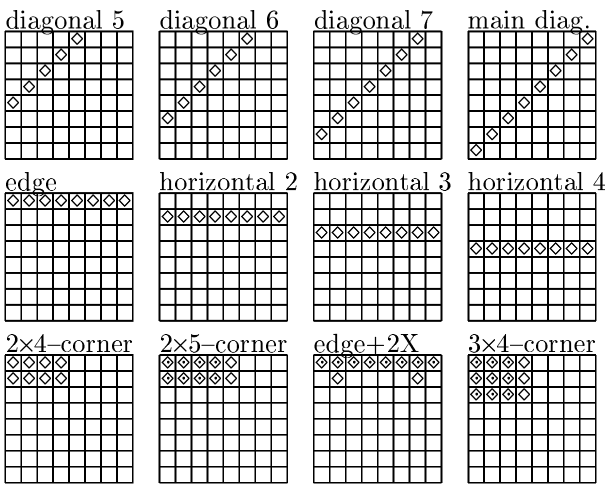


Figure 2: Used patterns (dots mark squares of sub–patterns which are described at the end of the section).

the position. On the other hand, averaging both values is more robust in this sense, increases the number of examples for each instance, since all occurrences contribute regardless of the side to move, and, finally, the table size can be reduced by roughly 50% due to color symmetry. Early experiments showed no advantage of the side–to–move distinction.

Another very important question is how the position sample space influences the estimates. Considering the application of the evaluation function, it seems reasonable to restrict the positions to those which actually occur in $\alpha\beta$ game–tree searches. Unfortunately, this approach doesn't work for the following reason:

Due to the nature of minimax searches, in the majority of visited positions the side to move is extremely favored. This effect causes a bias in the estimations since there appears to be an a priori winning probability much greater than 0.5, which is definitely not true for positions on principal variations. This bias introduces large evaluation oscillations in iterative deepening searches. Furthermore, while important patterns like edges get their deserved credit for correctly guessing the outcome after bad moves that — for example — give corners without compensation, the less important patterns suffer from extreme training examples in that their estimates are pushed towards 0 since these patterns simply don't matter in this case.

A pragmatic solution to both problems which gives good results is to take training positions from well–played games. This also has the advantage that all positions can be easily classified as a win, draw, or loss by negamaxing the tree built from the games. A disadvantage of this approach is that only a subset of the possible pattern instances occur in good games which may cause inaccurate evaluations when unknown instances appear during search. In extreme cases — for instance where corners are involved — a quiescence search can help, which explores lines more deeply in tactical situations.[5]

For an accurate estimation, a large number of examples is needed since the number of instances can be quite large. For example, the 8–disc pattern along the edge of the board has $3^8 = 6561$ instances. Fur-

[5] To overcome all these problems, currently a new approach is under investigation which gives each training position a weight that favors even positions. In this way tables are not only tuned for evaluating relevant positions on principal variations, but are also aware of rare situations if training positions generated by searches are also used.

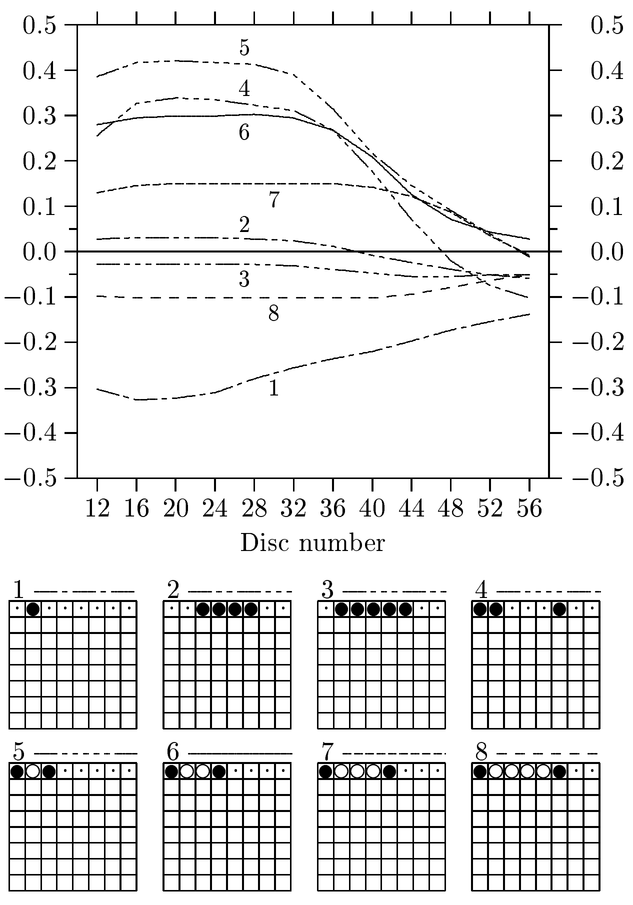


Figure 3: Values of some edge instances

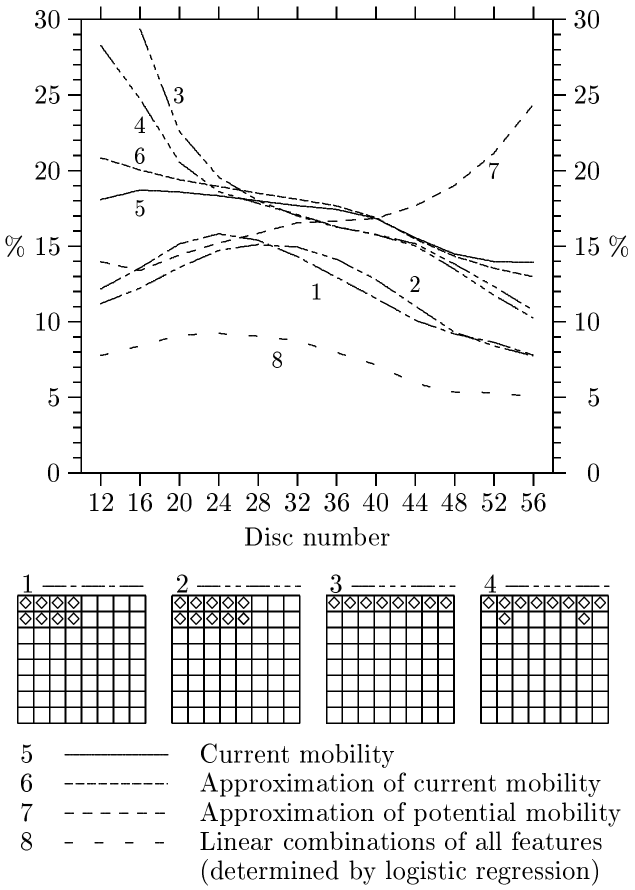


Figure 4: Discordance of important features and the evaluation function

thermore, pattern evaluations are normally game phase dependent, so even more examples are needed to estimate the table entries with a small variance. In order to mute this effect, it is possible to take advantage of pattern and color symmetry, as well as to smooth the estimates between adjacent phases.

Another problem caused by using well–played games lies in the local occurrence of pattern instances. For example, corners are normally occupied not until endgame, but even in the opening corner moves are examined in the course of minimax game–tree search and positions with occupied corners must be assigned a value. Therefore, it is reasonable to extrapolate evaluations to game phases in which pattern instances rarely occur.

As an application of the new pattern evaluation approach, Figure 3 shows the expected outcome for Black conditioned upon the occurrence of some interesting edge configurations after extrapolation and smoothing. The value for pattern p is estimated using the following expression:

$$V(p) = \frac{Y(p) + 0.5}{N(p) + 1.0} - 0.5$$

where $N(p)$ = number of positions containing p, and $Y(p)$ = number of positions containing p which are won for Black + $0.5 \cdot$ number of drawn positions containing p. The additive constants 0.5 and 1.0 assure a neutral evaluation (0) of pattern instances that do not occur.

While only good Othello players know the deep secrets of edge–play, some non–expert remarks are in order: A few games against experienced opponents suffice to teach a beginner that edge configuration 1 is very dangerous for Black because normally White can easily attack and win the upper left corner. Furthermore, the earlier that configuration appears on the board the more dangerous it is. Figure 3 underpins this observation. Configurations 2 and 3 regularly show up in tournament games, and examples 4–8 make clear that the simple concept "corners are good" must be qualified. Indeed, a short reflection reveals that there is a tradeoff between corner possession and mobility, and that while corner discs are the prominent stable discs which can be used as an anchor to produce more stable discs, at the end of the game there are more stable discs between them. These effects might explain the decreasing evaluation of configuration 4 and the evaluation differences of configurations 5–8.

Choosing Patterns When choosing patterns used for evaluation, one has to take into account their correlation, how well the table entries can be estimated given a certain number of example positions, how large the tables are, and how fast the according indices can be computed. The latter question suggests using lines of the board since their indices are already used for mobility approximation. On the other hand, experiments showed that game phase dependent 8–disc patterns are manageable even with the limitations of only about 80,000 available example games and a few million bytes of RAM for the tables. In order to compare the separation quality of different patterns, the example position set was divided into two parts. The first subset was used as a training set for pattern value estimation whereas the discordance was determined using the second set to avoid overfitting effects. A new very powerful 8–disc pattern, the 2×4–corner pattern, was found that even outperforms the widely used edge pattern, as shown in Figure 4.[6]

Recently two 10–disc patterns and one 12–disc pattern were incorporated in order to increase LOGISTELLO's knowledge about X–square—edge interaction and parity, namely the edge+2X, the 2×5– and the 3×4–corner pattern, which are shown in Figure 2. Since the number of examples was insufficient to estimate table entries for these large patterns for multiple stages of the game, single tables store the evaluation for all stages. Even with this restriction many configurations occurred rarely. Thus, the estimates of specific sub–configurations served as seed values using the following convex combination of estimates for the large configuration and its sub-configuration:

$$V(p) = \alpha \frac{Y(p) + 0.5}{N(p) + 1.0} + (1 - \alpha) \frac{Y(\text{sub } p) + 0.5}{N(\text{sub } p) + 1.0} - 0.5,$$

where $\alpha = \min\{1, N(p)/40\}$ is a weight which models the increasing influence of the large configuration estimate dependent on $N(p)$. In Figure 2, the squares of the chosen sub–patterns are marked with a dot.

Feature Combination

Often game stage dependent linear feature combinations are used to evaluate positions because they are quickly computable and there are efficient methods to determine the feature weights. Besides several hill climbing techniques three statistical approaches have been used:

- Mitchell (1984) labels Othello positions with the game result in the form of the disc–difference and performs a linear regression.
- Lee & Mahajan (1988) use a quadratic discriminant function assuming the features to be multivariate normal within the win and loss class.
- In (Buro 1994,1995) evaluation functions are described in the statistical framework of generalized linear models and logistic regression is used for parameter estimation.

These approaches give the evaluation function a game phase independent meaning. This is important if the evaluation function depends on the game phase and positions from different phases have to be compared

[6] An extensive pattern comparison can be found in (Buro 1994).

(e.g. selective extensions, reductions, or choosing opening book lines). Mitchell's evaluation function estimates the disc–difference at the end of the game, whereas Lee & Mahajan's quadratic discriminant function and logistic regression model the winning probability.

In order to determine LOGISTELLO's feature weights, the following statistical approaches were compared empirically:

- The quadratic discriminant function for normally distributed features with different covariance matrices within the win and loss class.
- The linear discriminant function for normally distributed features with equal covariance matrices (also known as Fisher's linear discriminant).
- Logistic regression, which makes no assumptions about the feature distribution.

Since each of these methods model the winning probability, they also apply to games without win degrees.

After maximum–likelihood parameter estimations for each model and several game phases defined by disc number, tournaments were played under usual timing conditions (30 minutes per player per game) using 100 nearly even starting positions from LOGISTELLO's opening book. It turned out that the weight vectors determined by logistic regression led to the strongest tournament program. A detailed description of these experiments can be found in (Buro 1994, 1995).

Implementation Aspects

In game–tree search there is a tradeoff between the evaluation function's quality and its computation time. Thus, it is very important to implement the function as efficient as possible once it has been chosen. In this section techniques are described which significantly increase the evaluation speed of the presented table–based evaluation function.

Incremental Index Updates

It is well known that incremental updates might be faster than performing all computation concerning evaluation at the leaves. With the table look–up approach it is necessary to determine indices from the current position. Assuming a 10×10 byte–array representation of the board — with +1 standing for black discs, 0 for empty squares, and −1 for white discs — the index representing the northern edge can be computed as follows:

```
3*(3*(3*(3*(3*(3*(3*b[11]+b[12])+b[13])+
   b[14])+b[15])+b[16])+b[17])+b[18]
```

At first glance, this expression seems to be very time–consuming. However, an optimizing compiler should be able to replace the multiplications by additions in order to speed up the computation. On average, the typical move flips 2–4 discs, depending of the stage of the game. Therefore, it would be faster to compute indices at the root position and to update them while making moves. Modifying one square's contents influences at most five line– and 2×4–corner–indices because diagonals of length less than four are not considered (Figure 5 gives an illustration). Thus, the following lines of C–code suffice to perform all necessary index updates caused by flipping or placing a single disc:

```
for (i=0; i < 5; i++)
  *(p->Index[i]) OP= p->OFFSET[i];
```

where `p` is a pointer to a pre–computed structure which contains the square's index addresses and index offsets, `OP` is either `+` or `-` depending on the side to move, and `OFFSET` is either `FlipOffset` or `PlaceOffset`. Here it is assumed that the compiler is able to unroll the loop.

Extending this scheme to the described 10– and 12–disc–patterns makes it necessary to increase the maximum number of index updates for some squares. It turned out that with LOGISTELLO's implementation this approach is slower than simply using the 2×4–corner–indices to compute the 2×5– and 3×4–corner–indices and using the edge–indices to determine the edge+2X–indices at the leaves.

Another important question is whether to perform copy/move or move/undo pairs in the tree–search process. On both Sun–Sparc and Intel–Pentium architecture LOGISTELLO's undo–move — that is flipping back discs, remove the placed disc, and update indices using negated offsets — is faster on average than to copy the entire board structure including indices.

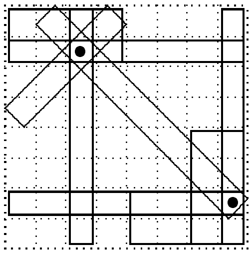


Figure 5: Indices influenced by squares c2 and h7.

Speed vs. Space — Table Combination

During the evaluation of the board, the indices for the diagonals, horizontals, and verticals are used three times: for the approximation of mobility and potential mobility, and for pattern instance evaluations. Rather than to access the three corresponding tables and to add the values after multiplication with the feature weights, it is possible to combine all three values into one table for each game phase in a preprocessing step. In this way, only one table has to be accessed for each line of the board and during evaluation multiplications with the feature weights are no longer needed. The same trick also applies to patterns which include other patterns. While this technique increases the space needed for the tables, it speeds up LOGISTELLO's search by about 75%.

Using a similar approach in their Othello program BILL, Lee & Mahajan (1990) were also able to speed up the evaluation by 'parallelizing' the computation of different features defined on the same set of patterns. However, their use of a quadratic feature combination made it impossible to combine *all* weighted table entries into a *single* value for each pattern instance.

Summary and Outlook

In this paper the evaluation function of one of today's strongest Othello programs has been presented together with the underlying concepts for the construction of features, their fast approximation, quality comparison, and combination. These approaches are almost entirely based on the analysis of a large set of example positions and do not require any manual parameter tuning.

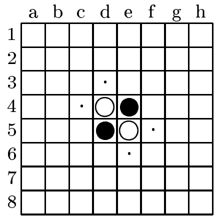
The general ideas — feature approximation, pattern instance evaluation, and feature combination, also apply to other games. For instance, in chess one might think of piece constellation patterns and the estimation of feature weights based on existing large game databases and logistic regression. However, some problems of pattern learning — like choosing the right sample space — are unresolved and still under investigation. Hopefully, a greater insight will lead to even stronger programs.

Acknowledgements

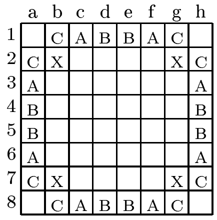
I am grateful to Mark Brockington and Warren Smith for their helpful comments on earlier versions of this paper.

Appendix: Othello

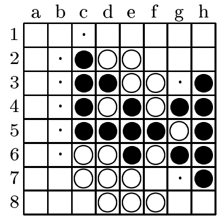
Othello is a two–person zero sum perfect information game played on a 8×8 board using 64 two–colored discs. Its rules are quite simple: Black goes first with White and Black alternating moves thereafter — if possible. In order to move, a disc is placed on an empty square showing the player's color such that the new disc and another own disc already on the board bracket at least one opponent's disc. All bracketed discs in all directions have to be turned, now showing the player's color. An example is given in the diagrams beneath. A player without legal moves has to pass. The game ends if neither player has a legal move in which case the player with the most discs on the board has won the game.



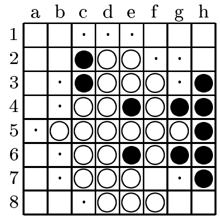
Start position



Square notation



White to move



Position after move b5

References

Buro, M. 1994. Techniques for the Evaluation of Game Positions Using Examples. *Ph.D. thesis (in German), University of Paderborn, Germany.*

Buro, M. 1995. Statistical Feature Combination for the Evaluation of Game Positions. *JAIR 3(1995): 373–382.*

Lee, K.F., Mahajan, S. 1988. A Pattern Classification Approach to Evaluation Function Learning. *Artificial Intelligence 36: 1–25.*

Lee, K.F., Mahajan, S. 1990. The Development of a World Class Othello Program. *Artificial Intelligence 43: 21–36.*

Mitchell, D.H. 1984. Using Features to Evaluate Positions in Experts' and Novices' Othello Games. *Master Thesis, Northwestern University, Evanston Illinois U.S.A.*

Rosenbloom, P.S. 1982. A World–Championship–Level Othello Program. *Artificial Intelligence 19: 279–320.*